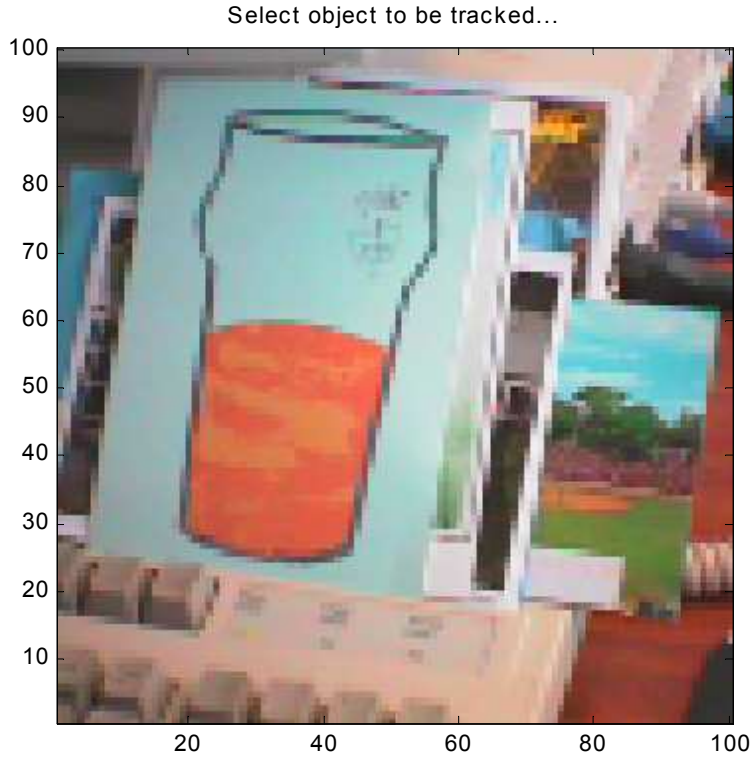
FireWire Vision Tools

A simple MATLAB interface for FireWire cameras



F. Wörnle, January 2008

Contents

1. Introduction 3
2. Installation 5
3. The FireWire Vision Tools interface 18
3.1 Supported video modes 18
3.2 Image capture commands 19
3.2.1 capImage 19
3.2.2 acqImage 22
3.3 Image processing commands 23
3.3.1 capProc 23
3.3.2 capClassify 28
3.3.3 imgProc 28
3.4 Selecting colour ranges 29
3.5 Simulink interface 30
3.6 Simple stereoscopic vision 34
Appendix A – The camera configuration file 36
Appendix B – Recompiling the drivers 37

1. Introduction

This document describes a small collection of MATLAB commands through which access is provided to digital cameras on the 1394 'FireWire' bus. Most of these commands are to be issued from the MATLAB command line but there also is an S-Function block which gives access to the camera from within a Simulink model.

Some of the commands of the toolbox include rudimentary image processing capabilities. It is possible to classify image content based on a programmable colour specification. For each detected object, the image processing algorithm returns the coordinates of the centroid as well as the coordinates of a boundary box around the object. Groups of objects sharing the same colour specification can be merged at different degrees, thereby allowing the granularity of the object detection system to be adjusted as required by the application.

The toolbox aims at providing an easy to use interface for student projects, laboratory experiments and any application with a need for simple machine vision, e. g. camera based robotics applications such as RoboCup, etc. In addition, a number of commands are based on the assumption that there are *two* cameras sharing a 1394 bus. This feature can be used in projects in which stereoscopic vision is required. All stereoscopic commands include the label 'Stereo' as part of their command name.

The toolbox should work well with most FireWire cameras. It is based on a universal driver for FireWire cameras: *'CMU 1394 Digital Camera Driver'* (version 6.4.4, www.cs.cmu.edu/~iwan/1394), developed at the *Robotics Institute* of *Carnegie Mellon University* (www.ri.cmu.edu). Please note that tests have exclusively been run with a *Unibrain fire-i* camera (www.unibrain.com), so it cannot be granted that other FireWire cameras will work – although this is highly likely, as both the CMU driver and the commands of the toolbox are fairly generic.

A variety of video modes are supported (YUV444, YUV422, YUV411, RGB8, Y8, Y16). Image dimensions can be defined freely from a minimum of 1 x 2 pixels up to the full size of a frame as specified by the selected mode (160 x 120 for YUV444, 320 x 240 for YUV422 and 640 x 480 for all other modes). Support of resolutions higher than 640 x 480 can easily be incorporated. However, this requires a few simple changes in the source code and recompilation.

The classification algorithms are those of the *Color Machine Vision project* (CMVision), a project conducted by the CORAL group at the *Carnegie Mellon School of Computer Science* (www.cs.cmu.edu). This software has been published under the GNU General Public License (GPL) and can be obtained from http://www-2.cs.cmu.edu/~jbruce/cmvision/ (CMVision project page). CMVision provides "a simple, robust vision system suitable for real time

robotics applications" by means of "global low level colour vision at video rates without the use of special purpose hardware".

To install the *FireWire Vision Tools* toolbox, extract the contents of the archive to a local folder. The "_bin" folder of the distribution should then be added to the MATLAB search path. The *CMU 1394 Digital Camera Driver* can be obtained from the above source. For convenience, the driver has also been included in the folder "_CMU_Driver_v644". Refer to section 2 for details of how to install this driver. Please note that this driver might have to be installed for **each** camera to be used with the toolbox (see section 2).

The *FireWire Vision Tools* are distributed as 'Free Software' under the terms of the GNU General Public License Agreement. Users are therefore given the right to copy, re-distribute and/or modify the source code to suit their needs.

The present release of this toolbox has been developed and tested using *MATLAB R2007a*. Comments and bug reports are always welcome. Please direct your feedback to:

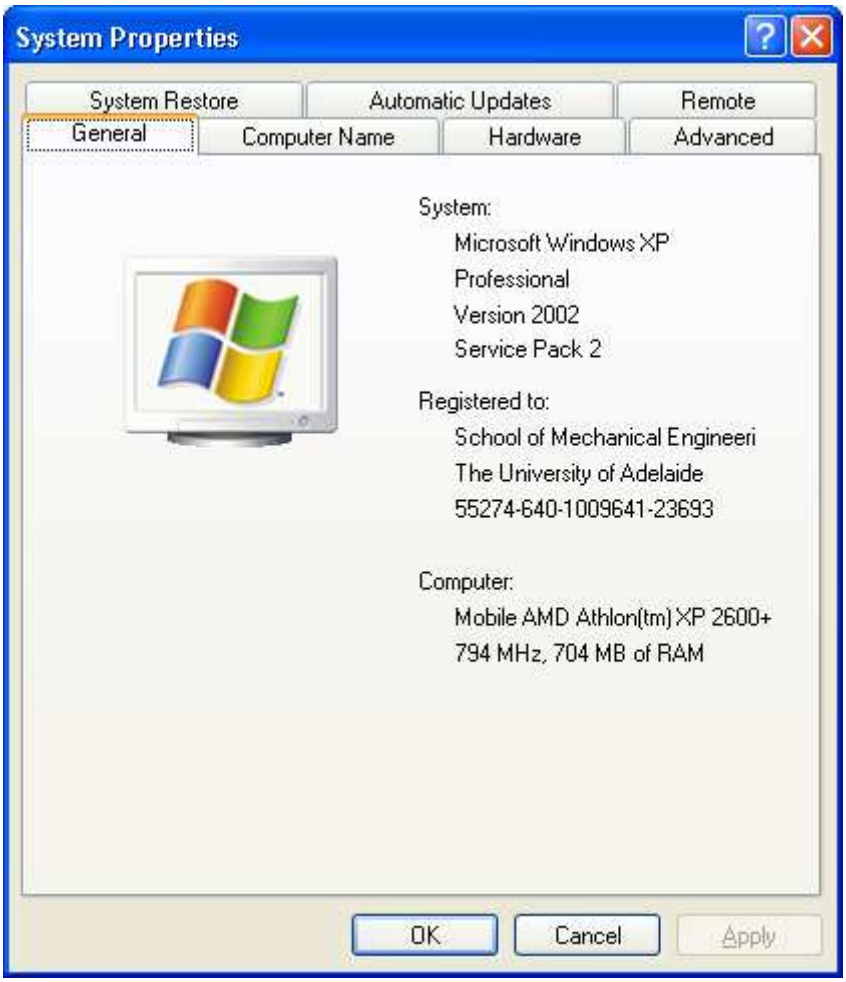
Frank Wornle (fwornle @yahoo.co.uk)

January, 2008

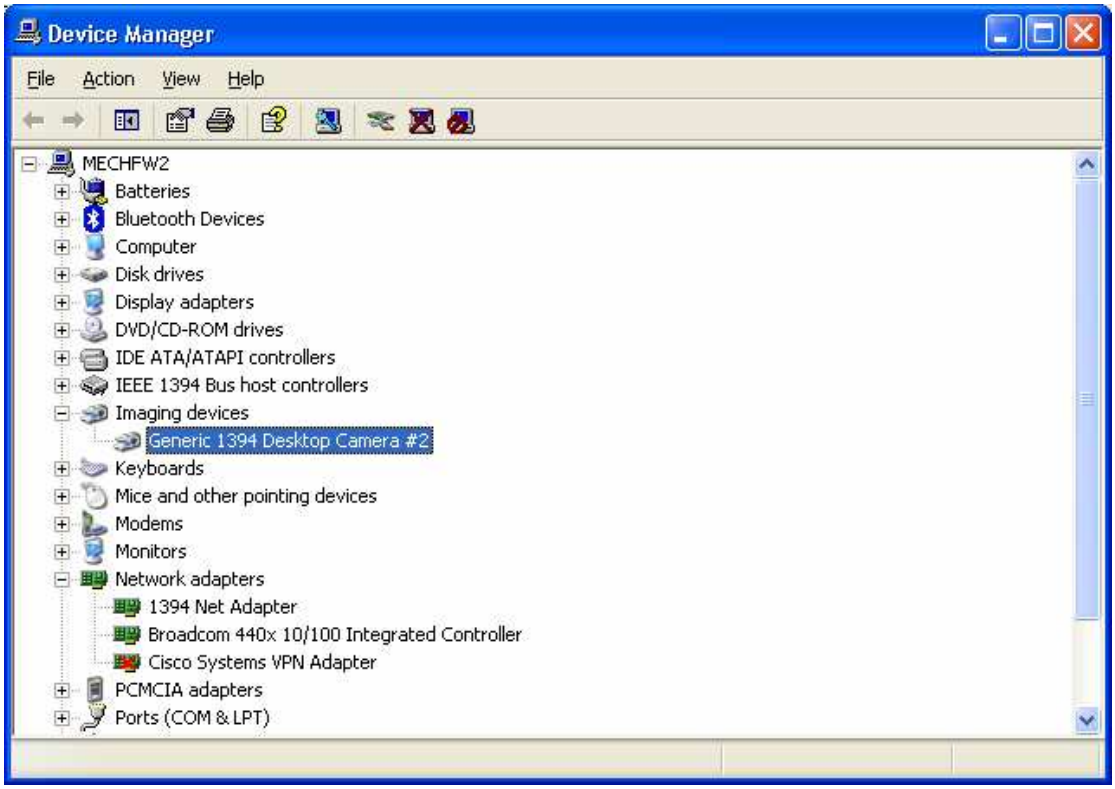
2. Installation

The *CMU 1394 Digital Camera Driver* can be installed from the folder "_CMU_Driver_v644" included in this distribution of the *FireWire_VisionTools* toolbox.

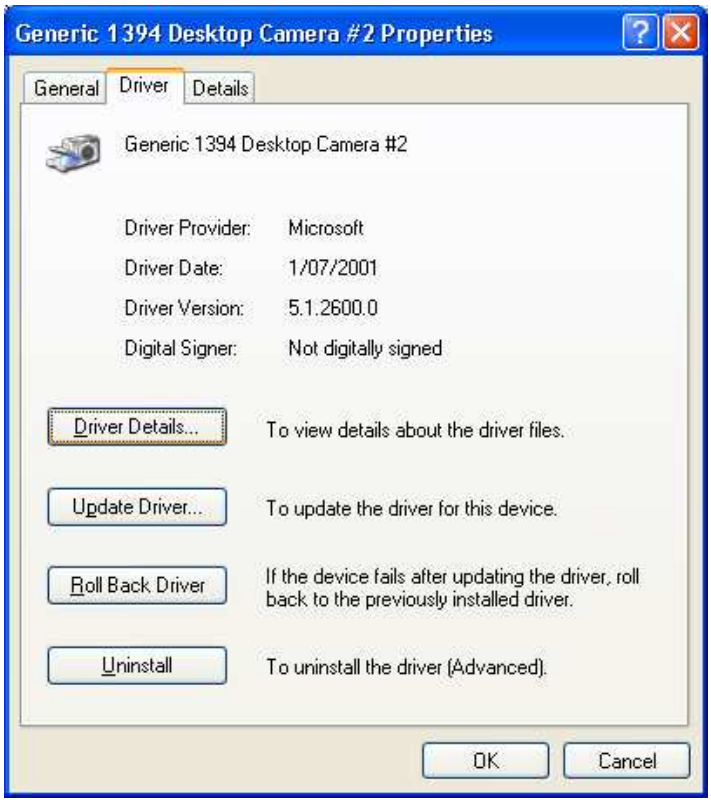
Connect the camera and open the *System Properties* from the *Control Panel* of *Windows start menu*.



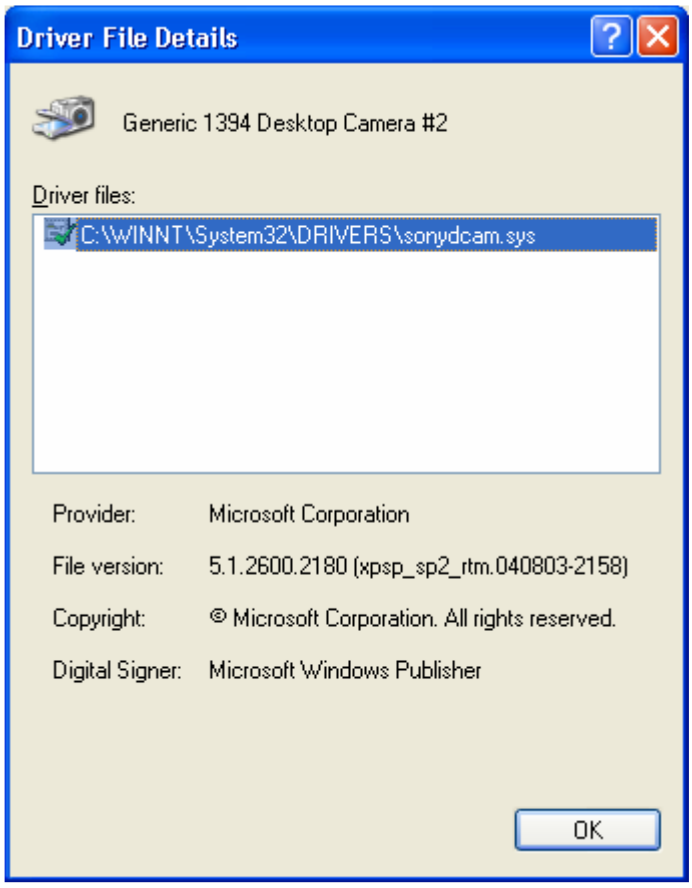
Select the *Hardware* tab and open the *Device Manager*. There should be an entry called *Imaging Devices* under which your camera will appear with its current name and driver settings.



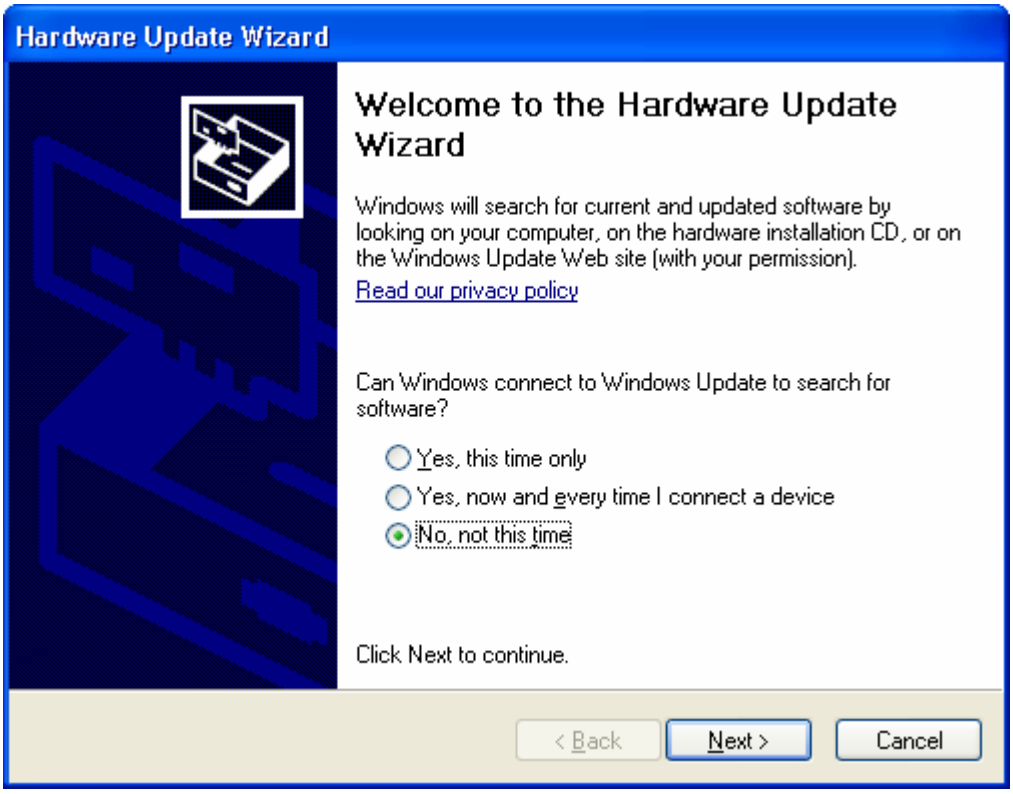
Double-click on the name of your camera to open its *Properties* control window and select the *Driver* tab.



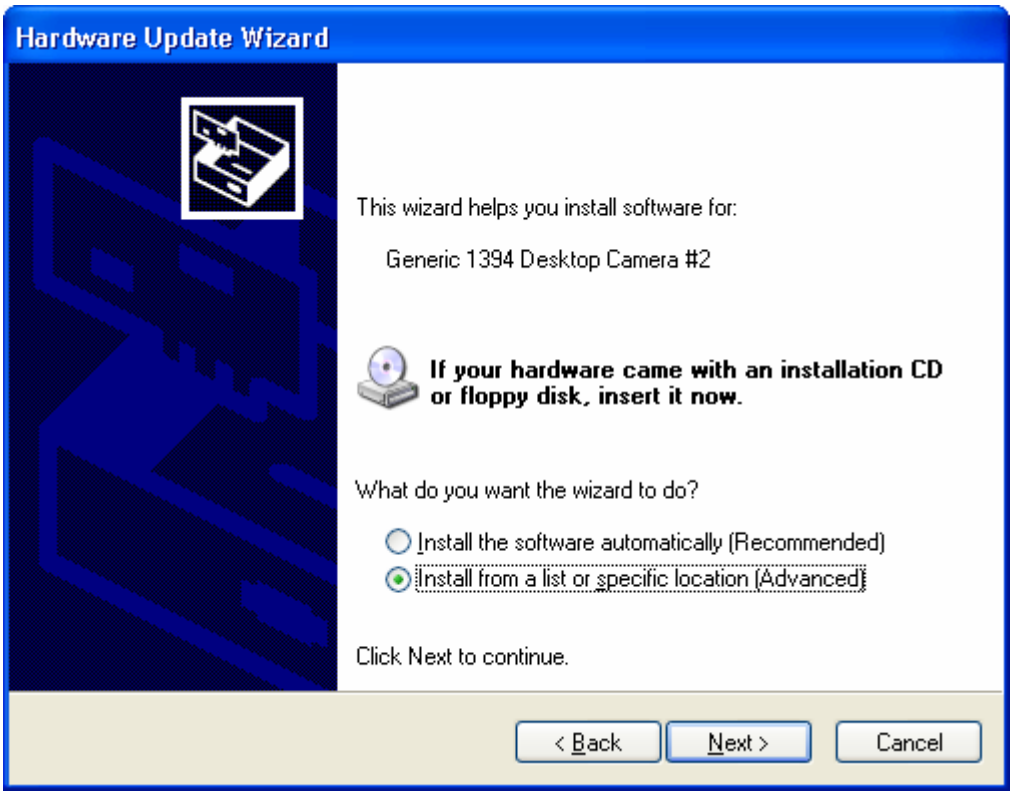
You can display the *Driver Details* to find out which driver is currently installed for this camera (e.g. the generic Windows driver *sonydcam.sys*).



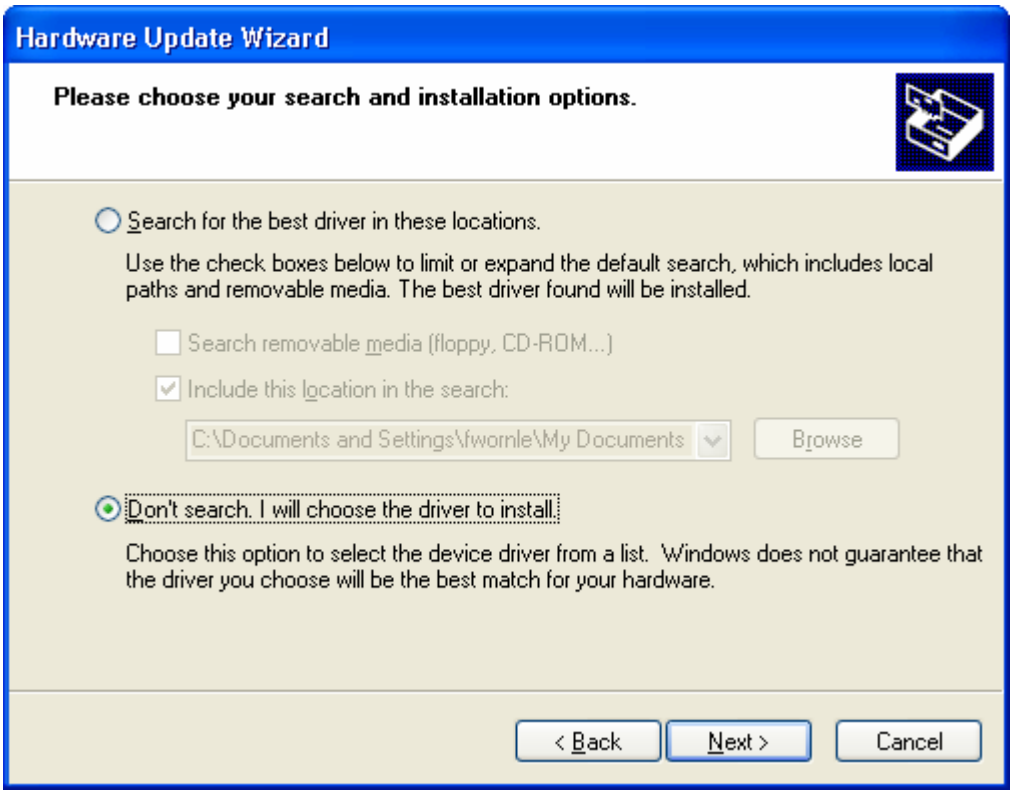
Close the *Driver details* window and click on *Update Driver*. This will launch the *Hardware Update Wizard*.


Hardware Update Wizard
Welcome to the Hardware Update Wizard
Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).
Read our privacy policy
Can Windows connect to Windows Update to search for software?
Yes, this time only
Yes, now and every time I connect a device
No, not this time
Click Next to continue.
< Back
Next >
Cancel

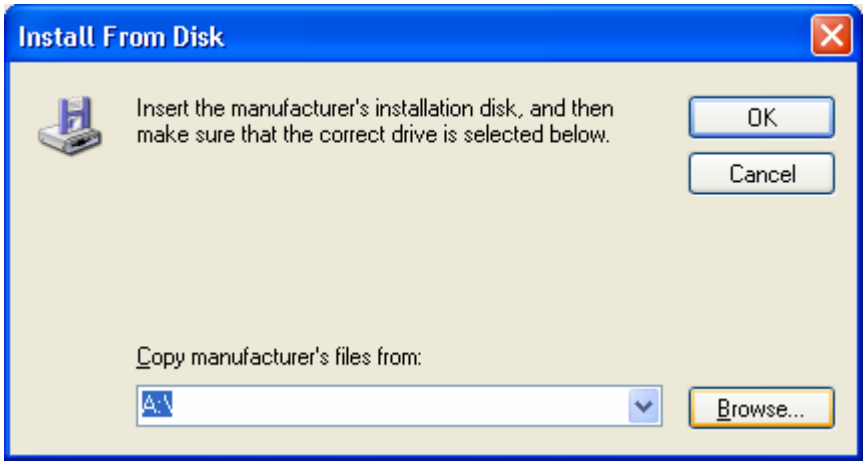
Choose *No, not this time* to prevent Windows from being overly eager in wasting your time. Click *Next*. On the next page choose *Install from a list or specific location (Advanced)* and click *Next*.


Hardware Update Wizard
This wizard helps you install software for:
Generic 1394 Desktop Camera #2
If your hardware came with an installation CD or floppy disk, insert it now.
What do you want the wizard to do?
Install the software automatically (Recommended)
Install from a list or specific location (Advanced)
Click Next to continue.
< Back
Next >
Cancel

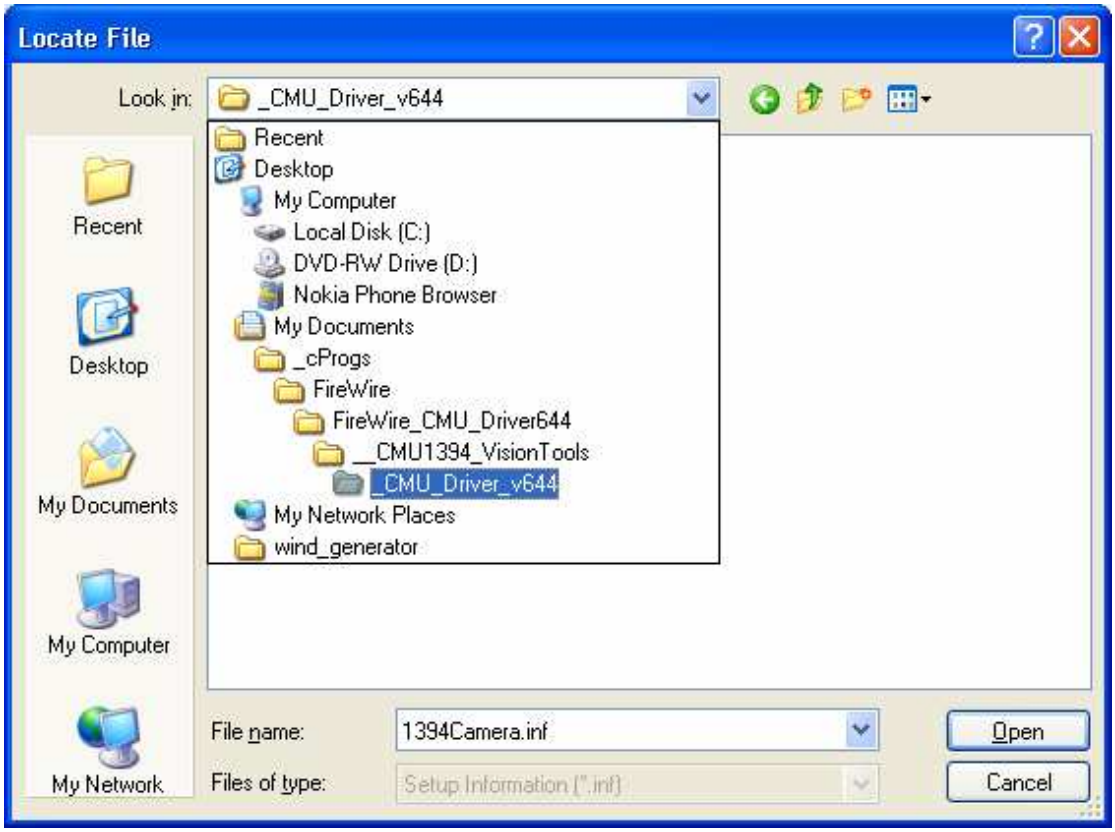
Select option *Don't search. I will choose the driver to install* and click *Next.*


Hardware Update Wizard
Please choose your search and installation options.
Search for the best driver in these locations.
Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.
Search removable media (floppy, CD-ROM...)
Include this location in the search:
C:\Documents and Settings\fwornle\My Documents
Browse
Don't search. I will choose the driver to install.
Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.
< Back
Next >
Cancel

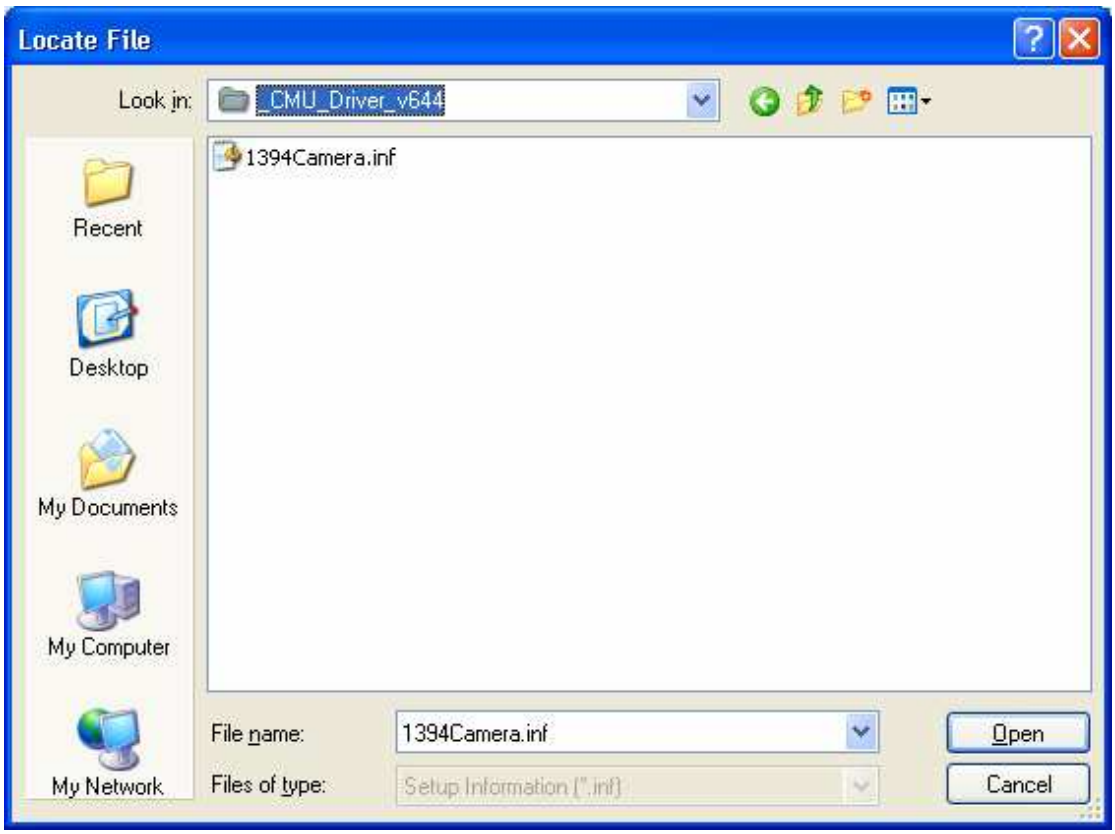
Windows might list a number of compatible drivers it knows of, but that is of no importance at this stage. Click on *Have Disk... .*



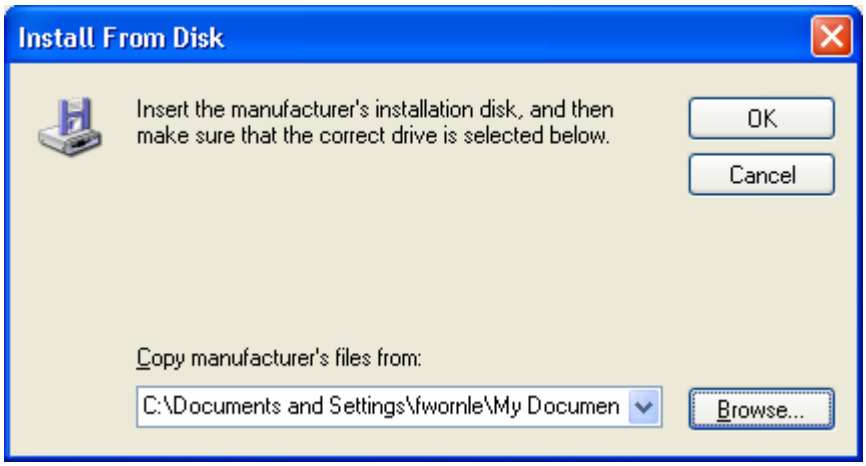
Click on *Browse…* and find the folder *_CMU_Driver_v644* included in the distribution of this toolbox.



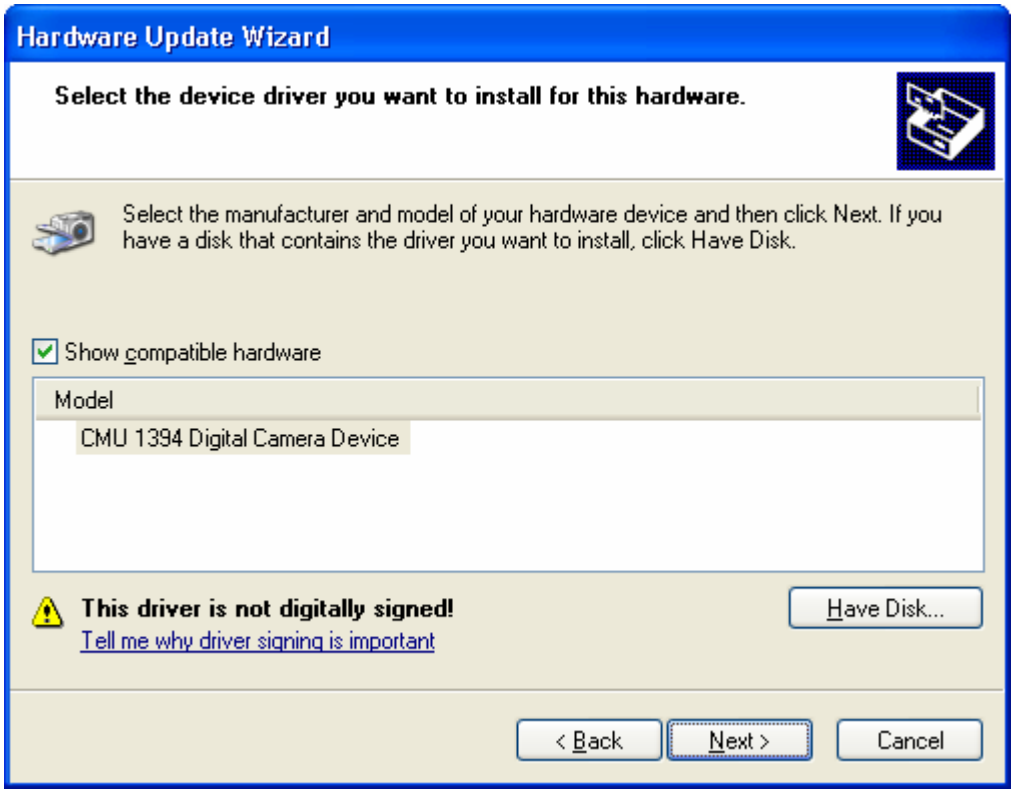
This folder should include the installation file *1394Camera.inf.*



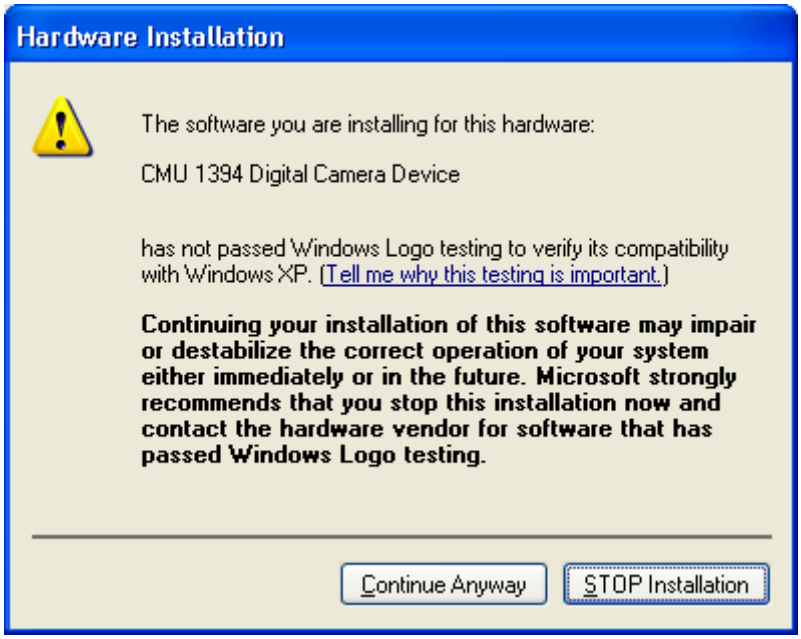
Click on *Open* and then click on *Ok*.


Install From Disk
Insert the manufacturer's installation disk, and then make sure that the correct drive is selected below.
OK
Cancel
Copy manufacturer's files from:
C:\Documents and Settings\fwornle\My Documen
Browse...

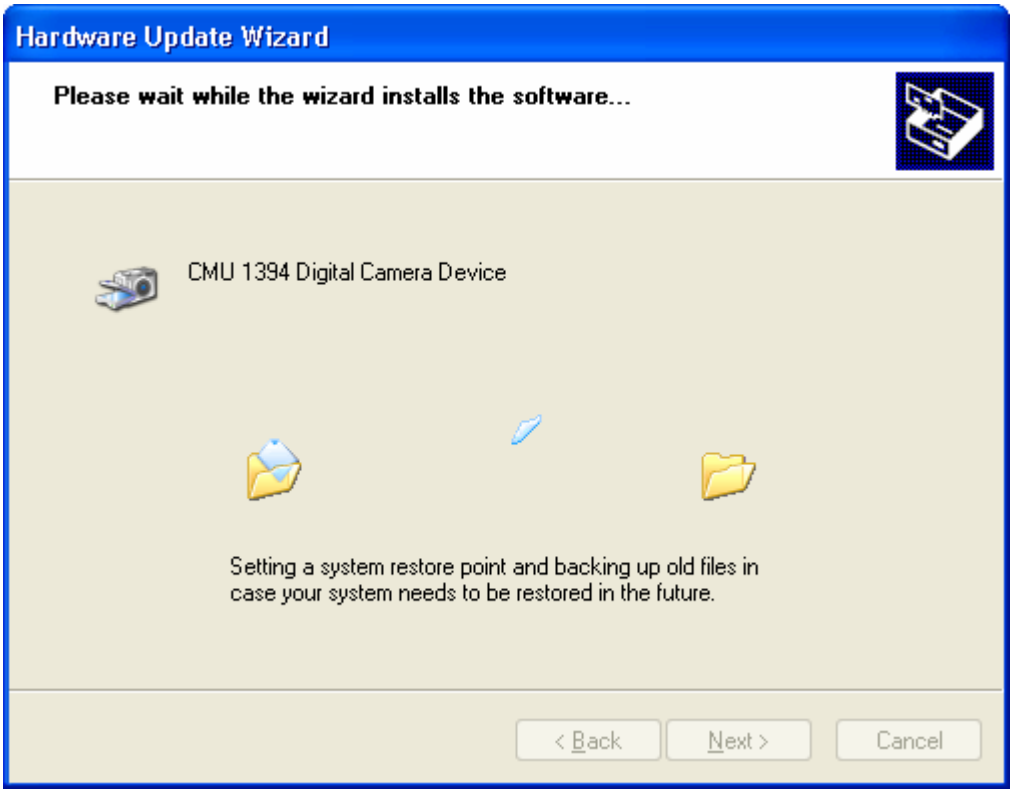
Windows will get all nervous and tell you that *This driver is not digitally signed!* which, as usual, can be ignored. Click on *Next*.


Hardware Update Wizard
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.
Show compatible hardware
Model
CMU 1394 Digital Camera Device
This driver is not digitally signed!
Tell me why driver signing is important
Have Disk...
< Back
Next >
Cancel

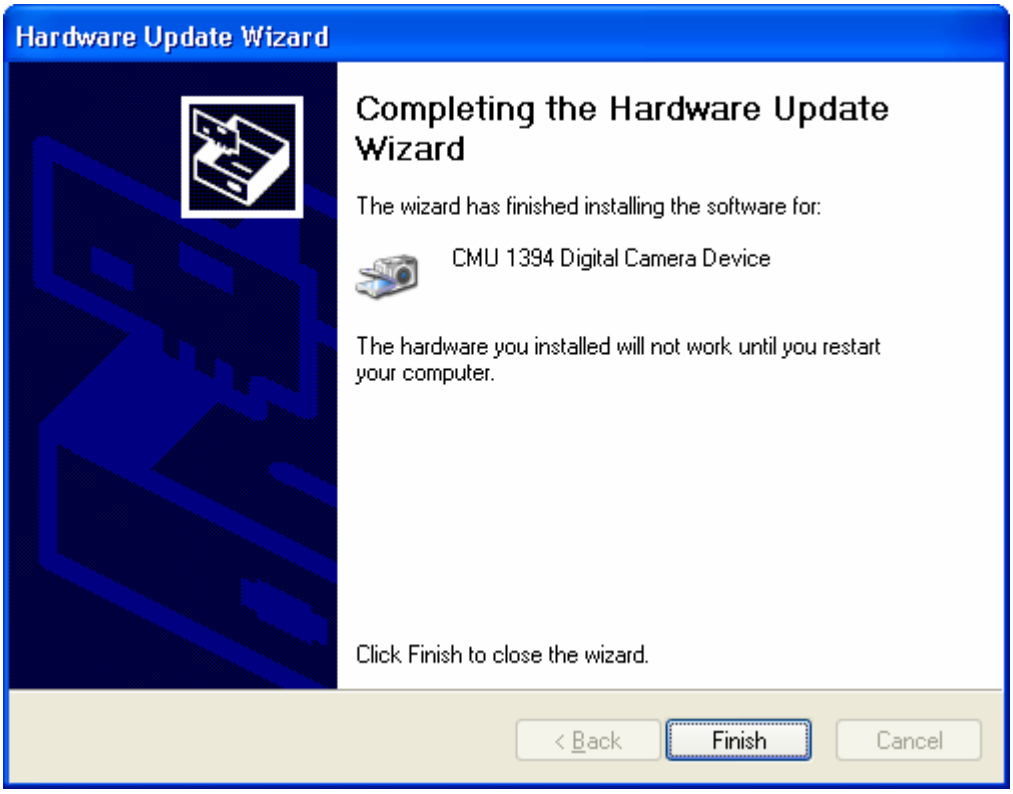
More waffle about the Microsoft Logo testing... click on *Continue Anyway*.


Hardware Installation
The software you are installing for this hardware:
CMU 1394 Digital Camera Device
has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)
Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.
Continue Anyway
STOP Installation

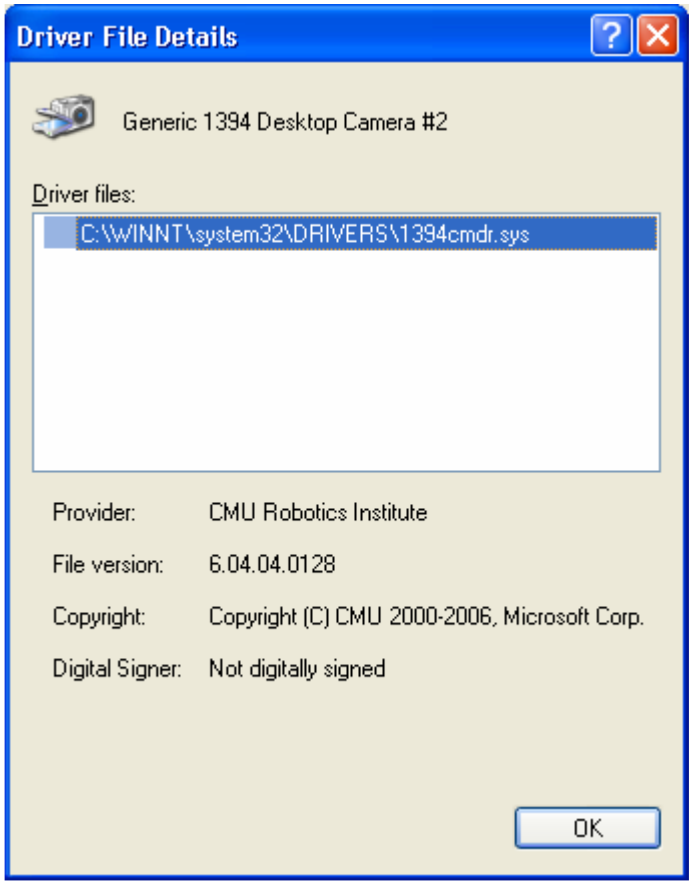
Now the actual driver file (*1394cmdr.sys*) gets copied to the Windows system32/DRIVERS folder and the corresponding registry entries are set.


Hardware Update Wizard
Please wait while the wizard installs the software...
CMU 1394 Digital Camera Device
Setting a system restore point and backing up old files in case your system needs to be restored in the future.
< Back
Next >
Cancel

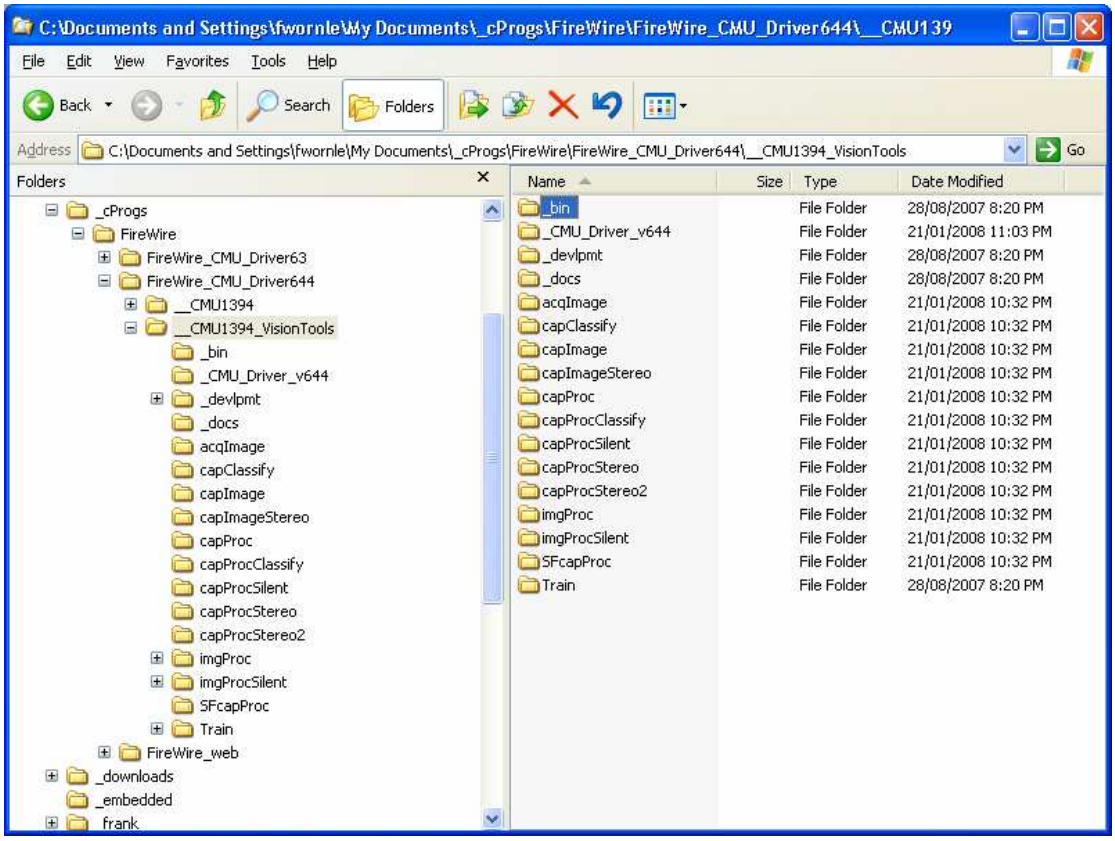
This concludes the installation of the *CMU 1394 Digital Camera Device* driver.


Hardware Update Wizard
Completing the Hardware Update Wizard
The wizard has finished installing the software for:
CMU 1394 Digital Camera Device
The hardware you installed will not work until you restart your computer.
Click Finish to close the wizard.
< Back
Finish
Cancel

The successful installation can be checked using *Driver Details…* button.



Once the driver has been installed, the toolbox is ready to be used. If not already done, add the "_bin" folder of the distribution to the MATLAB search path.



Each toolbox command comes with a simple test program, usually called "test". Change to the folder of a particular command, e.g. "acqImage" and start the m-script "test". Note that these scripts have been implemented as MATLAB functions, i. e. they can be called with call-up parameters:

To start the script with all parameters at their default values simply type

```
>> test
```

To test camera mode "1" (YUV422, 320 x 240) type

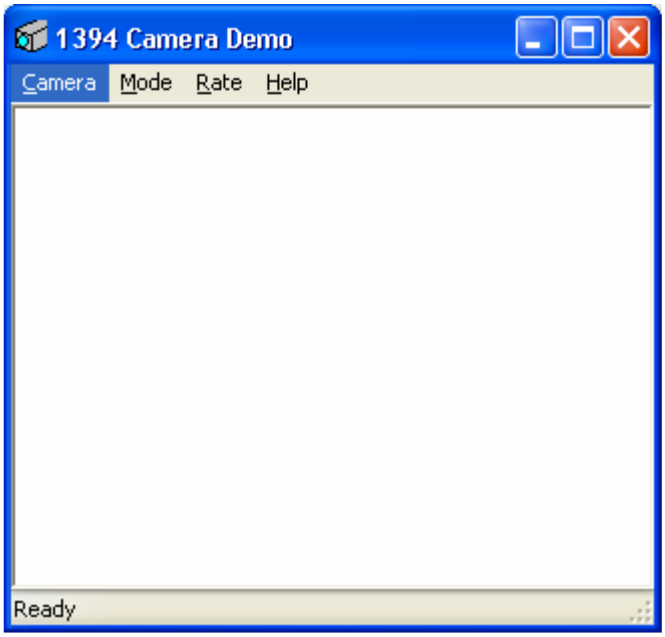
```
>> test(1)
```

To test the camera in mode "2" (YUV411, 640 x 480 – the default mode) while reducing the image size to 100 x 100 with the origin at (20, 50) type

```
>> test(2, 100, 100, 20, 50)
```

etc.

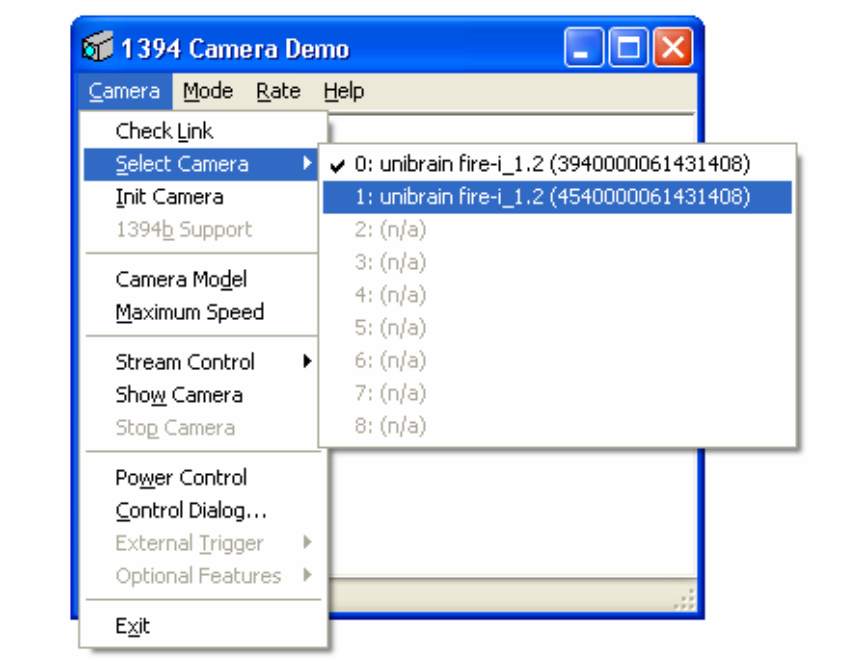
The available modes of a particular camera can be checked with the 1394 Camera Demo, which has also been included in this distribution (folder *_CMU_Driver_v644*). This small application can also be used to ensure the CMU driver has been installed properly for a particular camera (note: for some camera types the driver will have to be installed with each particular camera to be used with the system – for instance the *Unibrain fire-i* camera).



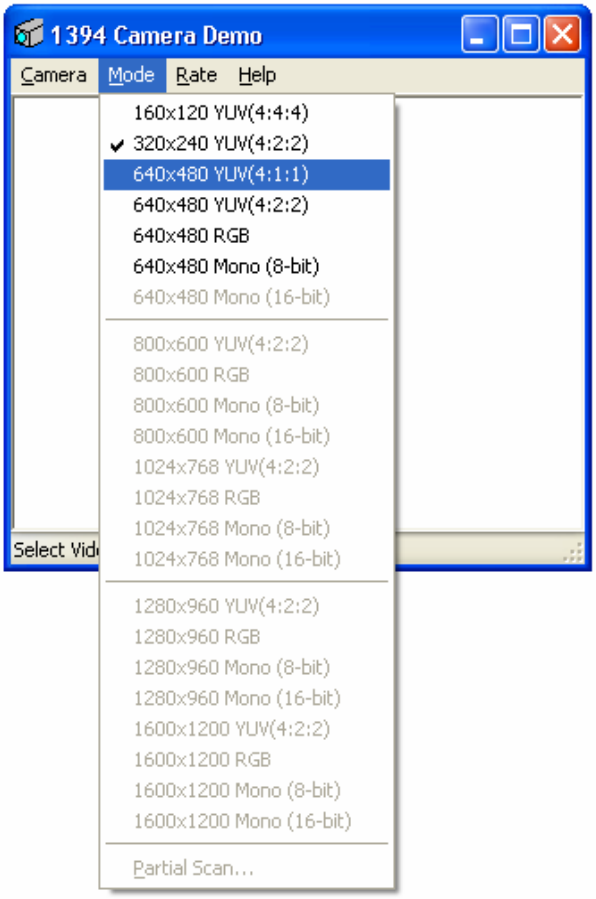
Check the link to the camera (menu: Camera). If a camera has been connected and is working properly the following message should be displayed:

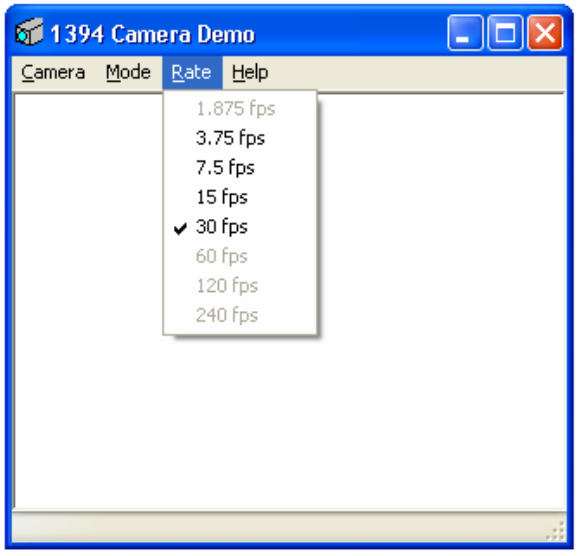
For stereoscopic vision two cameras need to be set up. Checking the link to the cameras will now indicated that two cameras have been found on the bus:

Each of these cameras has its own unique identifier – this is the reason why the driver has to be installed twice…



Once a camera has been selected and initialized, its features can be displayed using this application. For each mode, the maximum possible frame rate can be found this way. This is useful, if a toolbox command needs to be modified (and then recompiled) to match the capabilities of a camera which is not fully supported by the toolbox yet. However, for most cameras, this will not be required.


1394 Camera Demo
Camera
Mode
Rate
Help
160x120 YUV(4:4:4)
320x240 YUV(4:2:2)
640x480 YUV(4:1:1)
640x480 YUV(4:2:2)
640x480 RGB
640x480 Mono (8-bit)
640x480 Mono (16-bit)
800x600 YUV(4:2:2)
800x600 RGB
800x600 Mono (8-bit)
800x600 Mono (16-bit)
1024x768 YUV(4:2:2)
1024x768 RGB
1024x768 Mono (8-bit)
1024x768 Mono (16-bit)
1280x960 YUV(4:2:2)
1280x960 RGB
1280x960 Mono (8-bit)
1280x960 Mono (16-bit)
1600x1200 YUV(4:2:2)
1600x1200 RGB
1600x1200 Mono (8-bit)
1600x1200 Mono (16-bit)
Partial Scan...


1394 Camera Demo
Camera
Mode
Rate
Help
1.875 fps
3.75 fps
7.5 fps
15 fps
30 fps
60 fps
120 fps
240 fps

3. The FireWire Vision Tools interface

3.1 Supported video modes

Many FireWire cameras can be configured to produce digitized image information in a multitude of formats. The following video modes are supported by the commands of the *FireWire Vision Tools* toolbox:

- YUV444 Each pixel is represented by 3 bytes, namely the luminance (Y), and two bytes of colour information: C_r (= U) and C_b (= V). The camera stores these byte triplets in the following order U-Y-V.

- YUV422 Adjacent pairs of pixels are represented by 4 bytes, namely two individual luminance bytes (Y_1, Y_2), and two shared colour information bytes (U, V). The camera stores these bytes in the following order U-Y_1-V-Y_2.

- YUV411 Groups of 4 (adjacent) pixels are represented by 6 bytes, with four individual luminance bytes (Y_1 – Y_4), and two shared colour information bytes (U, V). The order of bytes is as follows: U-Y_1-Y_2-V-Y_3-Y_4.

- RGB8 Each pixel is represented by a colour triplet of three consecutive bytes: **R**ed, **G**reen, **B**lue.

- Y8 Only the luminance information is transmitted on the bus, with each pixel represented by one byte.

- Y16 Only the luminance information is transmitted on the bus. Each pixel is represented by two bytes.

The frame size and rate of an acquired image depends on the chosen video mode as well as a second ‘format’ parameter. This parameter can take the values ‘0’, ‘1’ or ‘2’, with frame sizes generally getting larger for larger values of ‘format’. Currently, the commands of the toolbox are configured for a fixed ‘format’ of ‘0’. This means that the frame sizes and rates are fixed as follows:

YUV444	160x120, 30 fps *)
YUV422	320x240, 30 fps
YUV411	640x480, 30 fps
YUV422	640x480, 15 fps
RGB8	640x480, 15 fps
Y8	640x480, 30 fps
Y16	640x480, 15 fps

*) fps = frames per second

Note:

(1) The *CMVision* colour detection algorithms work in the YUV422 colour space (= YUYV or UYVY). All commands which rely on *CMVision* therefore have to convert the acquired image data to this format and, if the image is returned to MATLAB, a second conversion to the output format (RGB8) is needed. On the other hand, YUV411 data consumes less bandwidth on the FireWire bus than YUV422 data and/or RGB8 data. The choice of the video mode should take into account these two constraints (conversion effort, bus bandwidth).

(2) For more details about image data formats and conversion routines to other formats see astronomy.swin.edu.au/~pbourke/dataformats/yuv.

3.2 Image capture commands

This section describes commands which can be used from the MATLAB command line and/or m-scripts to acquire visual information and return it to the workspace in form of a cell array.

3.2.1 capImage

This command acquires a single frame from the camera. The call-up syntax is as follows:

```
>> capImage
??? Usage:   data = capImage(mode [, verbose][, sizeX, sizeY][, originX, originY])
    mode:    -1 STOP
              0 YUV444 (160x120, 30 fps)
              1 YUV422 (320x240, 30 fps)
              2 YUV411 (640x480, 30 fps)
              3 YUV422 (640x480, 15 fps)
              4 RGB8   (640x480, 15 fps)
              5 Y8     (640x480, 30 fps)
              6 Y16    (640x480, 15 fps)
    verbose:  0 silent
              1 verbose
```

Like all commands of the toolbox, 'capImage' requires specification of the video ***mode*** (0 – 6). In addition to this compulsory call-up parameter, there are up to 5 optional parameters: a ***verbose*** flag, the ***size*** of the section of the acquired frame to be returned as well as the ***origin*** of this section within the frame.

Setting *verbose* to '1' causes the camera settings to be written to the MATLAB command window. This can be useful when the exact features of a particular FireWire camera are not known, e.g.:

```
>> image(capImage(2,1))

Adjusting camera settings...
Programming camera '1' with: 6 90 96 69 187 511 304
done.

Number of cameras found:       1
Device description (node 0):   unibrain fire-i_1.2 (4540000061431408)
Camera name:                   Fire-i 1.2
Vendor name:                   Unibrain
Version number:                256
Max speed:                     400
Link status                    0
Has power control:             1
Status power control:          0
Has 1394b:                     0
Status 1394b:                  0
Number of channels (camera):   0
Current channel (camera):      0
Camera supports video mode:    160X120 YUV 4:4:4 [24 bits per pixel]
Camera supports video mode:    320X240 YUV 4:2:2 [16 bits per pixel]
Camera supports video mode:    640X480 YUV 4:1:1 [12 bits per pixel]
Camera supports video mode:    640X480 YUV 4:2:2 [16 bits per pixel]
Camera supports video mode:    640X480 RGB (8-bit) [24 bits per pixel]
Camera supports video mode:    640X480 Mono (8-bit) [8 bits per pixel]
Unsupported video mode:        640X480 Mono (16-bit)

Feature 'Brightness' (ID 0)
Range (min, max):              128, 383
Current value:                 304
Attribute 'AutoMode':          0

Feature 'Auto Exposure' (ID 1)
Range (min, max):              0, 511
Current value:                 511
Attribute 'AutoMode':          0

Feature 'Sharpness' (ID 2)
Range (min, max):              0, 255
Current value:                 80

Feature 'White Balance' (ID 3)
Range (min, max):              0, 255
Current values (low, high):    96, 69
Attribute 'AutoMode':          0

Feature 'Saturation' (ID 5)
Range (min, max):              0, 255
Current value:                 90

Feature 'Shutter' (ID 7)
Range (min, max):              0, 7
Current value:                 6

Feature 'Gain' (ID 8)
Range (min, max):              0, 255
Current value:                 187
```

The displayed status information reveals for example that, on the *fire-i* camera, there is no 16-bit black-and-white video mode (Y16). Piping the output of the 'capImage' command through to the 'image' command of MATLAB is an easy way to display the acquired image.

Alternatively, the image data can be assigned to a MATLAB workspace variable. Independent of the chosen video mode, the output format is always an *n x n x 3* cell array containing the corresponding Red, Green and Blue values.

The following lines cause MATLAB to continuously acquire YUV411 images, convert them to RGB and display them on screen. The program stops after 100 frames. Upon completion of 100 cycles, image capturing is stopped by calling the driver with call-up parameter '-1'.

```
% start camera, YUV411 mode
for(i=1:100)
   image(capImage(2));
   drawnow
end

% stop camera
capImage(-1);
```

Note:

Fixed camera settings are expected to be defined in a text file called *cameraconfig.txt*. This file should be located in the folder the capturing command is called from. If *cameraconfig.txt* cannot be found, the camera is switched back to automatic mode.

The following six hardware attributes of the camera can be set in *cameraconfig.txt.* The specified values are examples which gave good results in our laboratory at the University of Adelaide (indoors environment, bright):

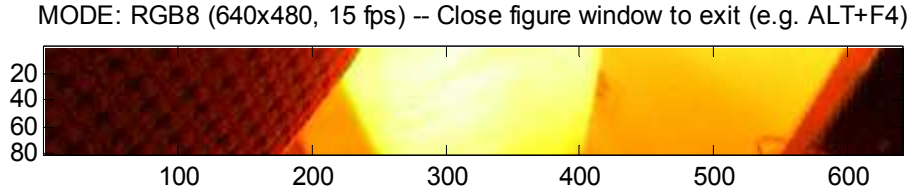
Shutter speed:	6
Saturation level:	90
White balance:	96, 69 (fixed → automatic adjustment switched off)
Gain:	87
Auto exposure:	511 (fixed → automatic adjustment switched off)
Brightness:	304 (fixed → automatic adjustment switched off)

3.2.2 acqImage

This command does much the same as 'capImage' (section 3.2.1). However, internally images are taken from a constant stream of images. This may (or may not) result in a slightly better performance.

Note that, as for all other commands, the size of the image to be returned can be reduced. This may be useful, when the horizontal position of an object is to be determined (e.g. far left, left, centre, right, far right). In this case, it might be much quicker to just process a reduce part of the captured image – possibly just a few lines and/or columns.

In particular, it might be much more efficient to detect an object (e.g. a mobile robot on the floor) in an initially fully sized frame and then "home in" on the detected object for tracking. This way, the bulk of the image processing is done on a much smaller number of pixels than what would be the case when processing a full sized frame. Every time the robot reaches the limits of the reduced area, its origin is adjusted by a small Δx and Δy to re-adjust the relative position of the robot to the centre of the processing frame. This approach should lead to a much improved overall performance of the image processing system.

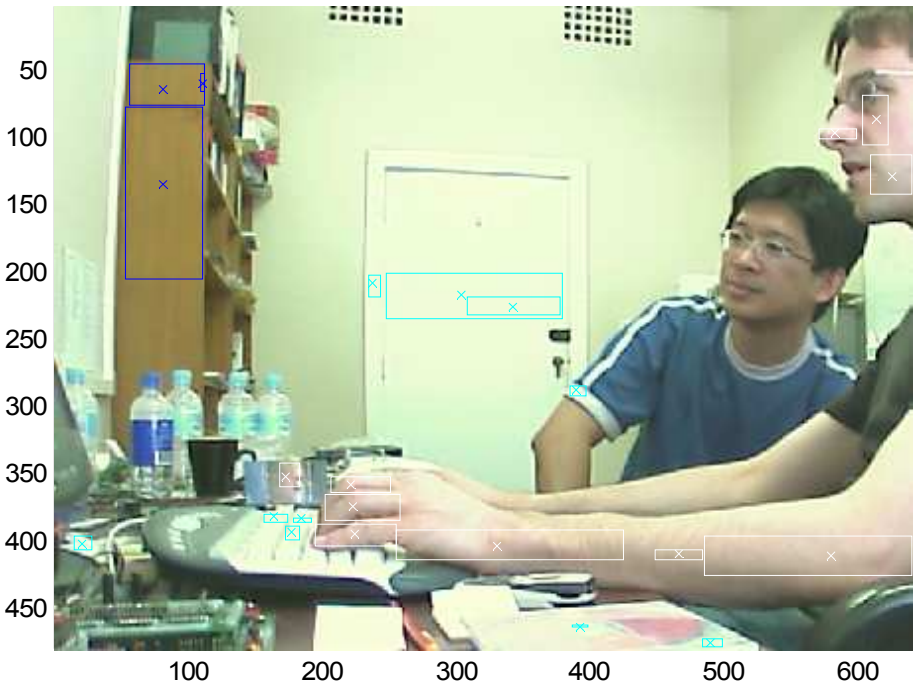


3.3 Image processing commands

The commands presented in this section combine image acquisition (e.g. capImage) with the data classification algorithms of the CMVision project (see: The CORAL group, www-2.cs.cmu.edu/~jbruce/cmvision/).

3.3.1 capProc

This command can be used to detect and track objects. A number of CMVision algorithms are used to classify the acquired image data. *Clusters* with the requested size and/or colour qualities are being detected and made available to MATLAB in form of a *cluster information structure*. This structure includes, among other things, the position and size of a rectangular box around the cluster as well as the coordinates of its centroid. The location of the centroid can directly be used in tracking applications.



Detecting 3 different colours - Cluster *boundary box* and cluster *centroid*

The above example demonstrates the situation for a relatively selective colour specification, i. e. with rather narrow ranges in each category Y, U, V. A boundary box has been superimposed with the captured image to reveal the

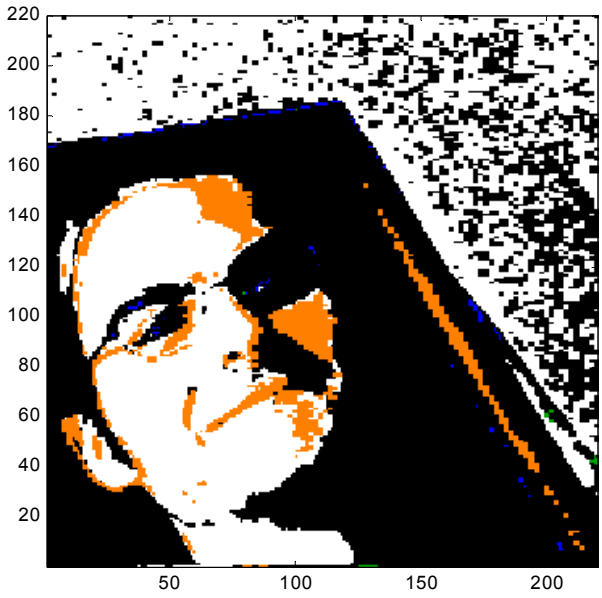
size of the detected cluster. The small star denotes the location of the centroid.

The admissible colour ranges are communicated to the image processing engine by means of a colour definition file (e.g. 'color.txt'). This text file can include colour attributes for up to 32 different objects to be detected. The general structure of the colour file is shown below:

```
[Colors]
(255,128,  0) 0.5000 2 Ball
(255,255,  0) 0.6000 3 Yellow_Team
(  0,  0,255) 0.6000 3 Blue_Team
(255,255,255) 0.0000 0 White
(255,  0,255) 0.0000 0 Marker_Pink
(160,  0,160) 0.0000 0 Marker_Purple
(  0,160,  0) 0.0000 0 Marker_Green
(  0,  0,  0) 0.0000 0 C08
(  0,  0,  0) 0.0000 0 C09
(  0,  0,  0) 0.0000 0 C10
(  0,  0,  0) 0.0000 0 C11
(  0,  0,  0) 0.0000 0 C12
(  0,  0,  0) 0.0000 0 C13
(  0,  0,  0) 0.0000 0 C14
(  0,  0,  0) 0.0000 0 C15
(  0,  0,  0) 0.0000 0 C16
(255,128,  0) 0.0000 0 Ball_2
(255,255,  0) 0.0000 0 Yellow_Team_2
(  0,  0,255) 0.0000 0 Blue_Team_2
(255,255,255) 0.0000 0 White_2
(255,  0,255) 0.0000 0 Marker_Pink_2
(160,  0,160) 0.0000 0 Marker_Purple_2
(  0,160,  0) 0.0000 0 Marker_Green_2


[Thresholds]
(  7:175, 50:150,160:200)
( 47:120,  5:80, 130:200)
( 76:112,110:190, 67:128)
(130:255, 81:131,125:178)
( 50:181,102:135,190:222)
(103: 96,118:140,144:166)
( 67:134, 96:129, 85:124)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
( 86:221, 35: 79,102:150)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
(  0:  0,  0:  0,  0:  0)
```

Section **[Colors]** includes four columns:

(1) A user defined RGB colour triplet which can be used to visualize the pixels of a detected cluster. This can be useful to validate the settings of a particular colour detection range. An example with four different detected regions is shown below.



Setting all pixels of a detected cluster to its default colour

(2) The *merge density parameter* is assigned a value between 0 and 1. It it is a measure for the degree of object merging which takes place upon colour detection. A value near '1' results in many separate small objects (very little merging between the individual clusters of a particular colour). Decreasing the *merge density parameter* allows for an increasing amount of merging to take place, thereby resulting in fewer but bigger objects. This parameter thus controls the granularity of detected clusters within an acquired image.

(3) Colour ID; this parameter can be used to identify clusters of a particular colour quality.

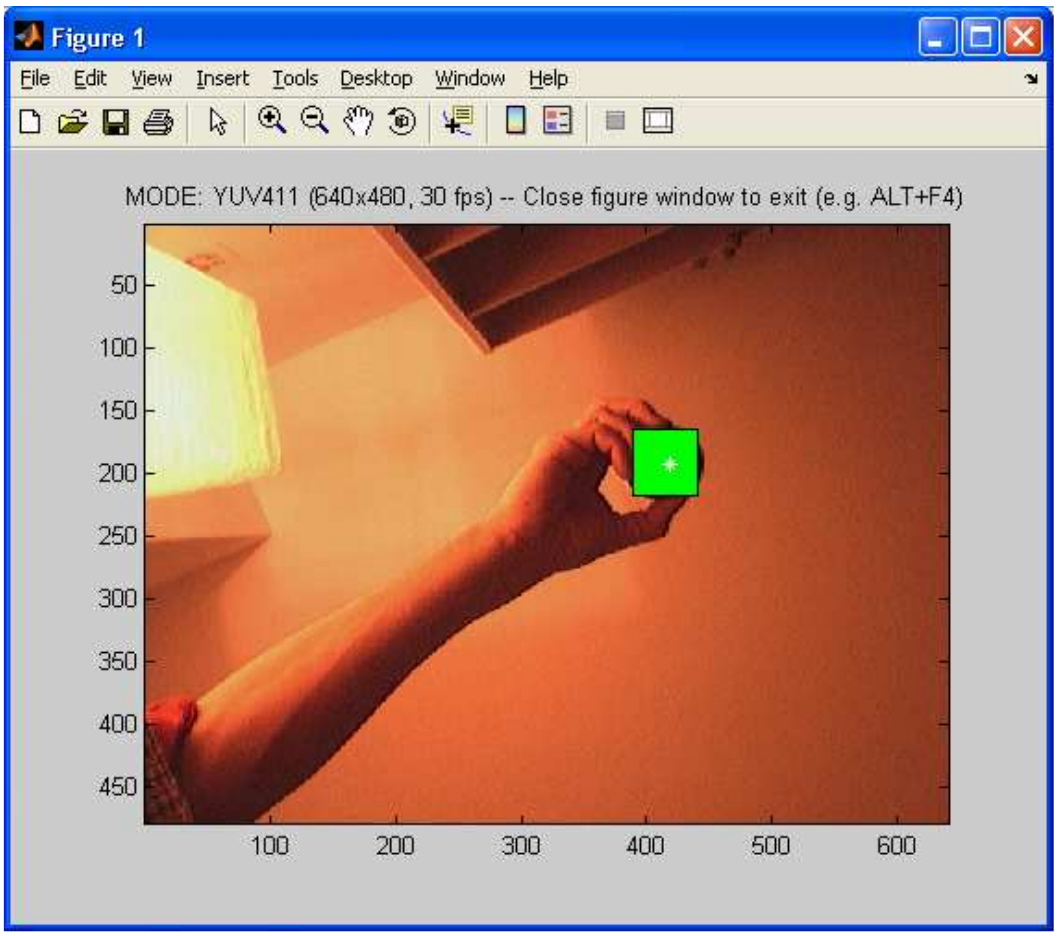
(4) Colour name; this optional name can be used to link a detected object to a clear text identifier.

Section **[Thresholds]** contains the RGB colour ranges which are used to detect a particular cluster.

For further details about the CMVision algorithms, please refer to the CMVision manuals (www-2.cs.cmu.edu/~jbruce/cmvision/).

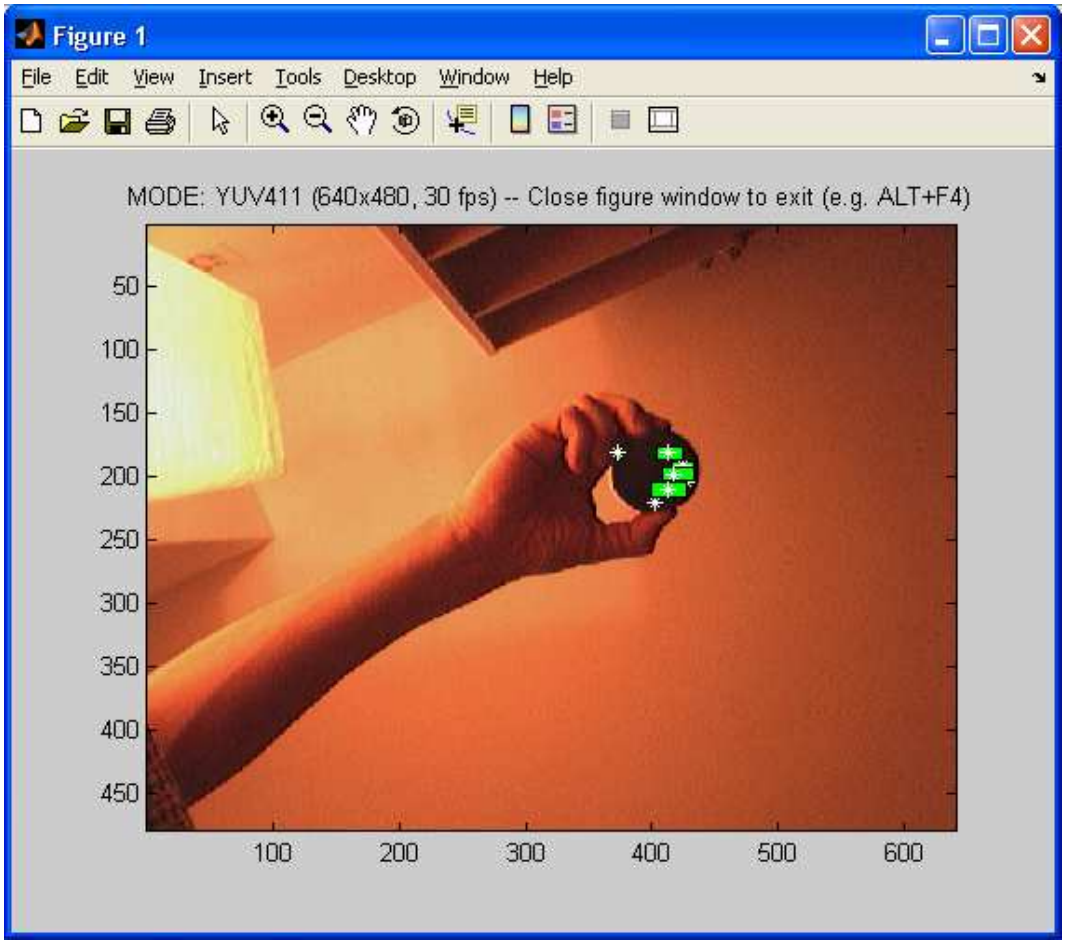
For example, the following colour specification was used to produce the output shown below

```
[Colors]
(0,   255,   0)  0.15  1  green

[Thresholds]
(12:24, 124:128, 125:133)       % black ball, living room, night
```



On the other hand, setting the *merge density parameter* to 0.95 produces a much more fine granular result:

```
[Colors]
(0,   255,   0)   0.95  1  green

[Thresholds]
(20:32, 121:125, 130:138)       % black ball, living room, night
```



capProc can be invoked with the following call-up parameters:

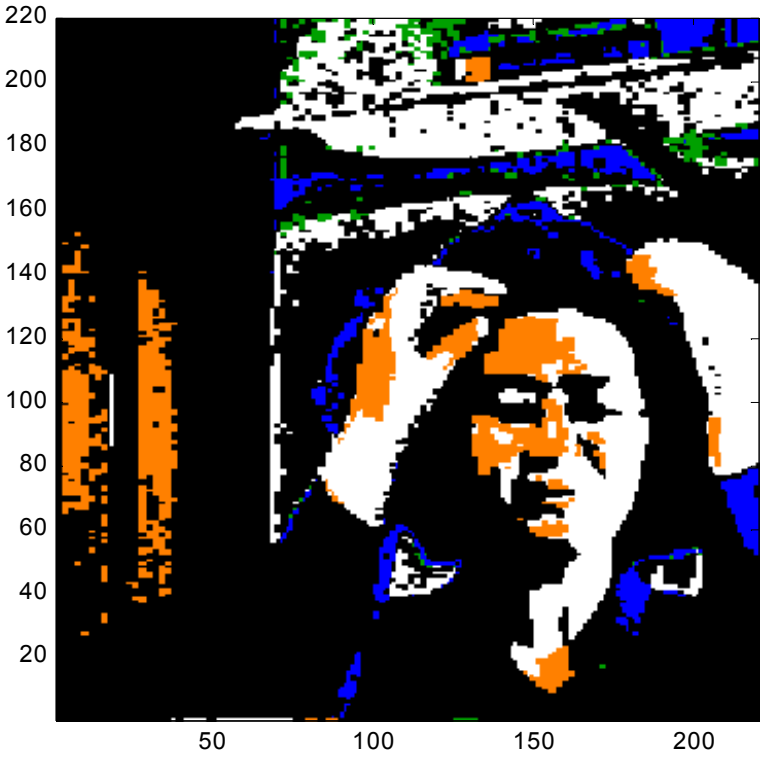
```
[Region_StructureArray [,RGB_Data]] = capProc(   mode
                                                 [, 'colour_filename']
                                                 [, colour_list]
                                                 [, verbose]
                                                 [, sizeX, sizeY]
                                                 [, originX, originY]  )
```

As with *capImage*, the only required parameter is the video ***mode***. If the (optional) ***colour_filename*** has not been specified, the default name "testcolours.txt" is used. In addition to these call-up parameters, a ***verbose*** flag, the ***size*** of the image to be processed (and optionally returned) as well as the ***origin*** of this subsection within the entire frame can be specified.

capProc always returns the result of the image processing step (clusters, centroids, colours, names, ...). Additionally, the corresponding image data can be returned in an optional 2nd output parameter (RGB).

3.3.2 capClassify

The command *capClassify* can be used to find appropriate settings for a particular parameter in the colour file. All pixels of a detected cluster are set to their default colour (1st column in section [Colors]).



3.3.3 imgProc

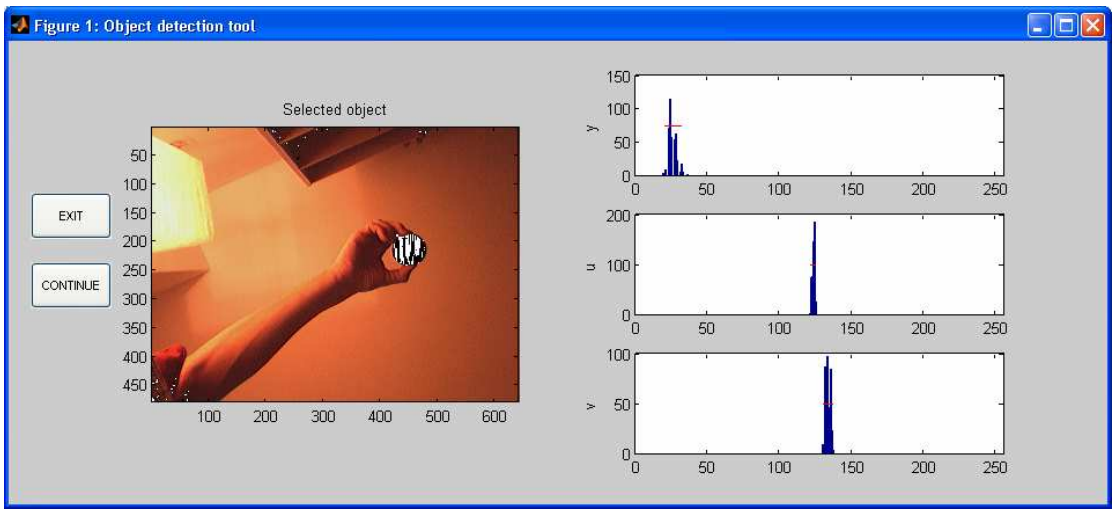
The command *imgProc* can be used to test the validity of a chosen colour definition. This command takes a previously stored RGB image as input and processes it using CMVision. Optionally, the ***colour filename*** and/or a vector of ***colour IDs*** can be specified. The command returns a list of all detected regions together with their size, co-ordinates of the centroid, etc. See the test program in folder *imgProc* for further details.

3.4 Selecting colour ranges

Object tracking by colour detection relies on the correct definition of appropriate RGB (or YUV) colour ranges. Too wide a range will lead to an unwanted detection of objects with similar colour qualities; on the other hand, too narrow ranges might cause the algorithm to return without having detected anything.

To determine appropriate RGB ranges, a number of MATLAB m-file scripts have been provided. These 'training' programs allow a user to select an object to be tracked from either a still image (*trainStill.m*) or a live stream of images (*trainCamera.m*). Both programs return the colour characteristics of the selected object. The thus established values (ranges of Y, U, V) can directly be copied to a colour definition file.

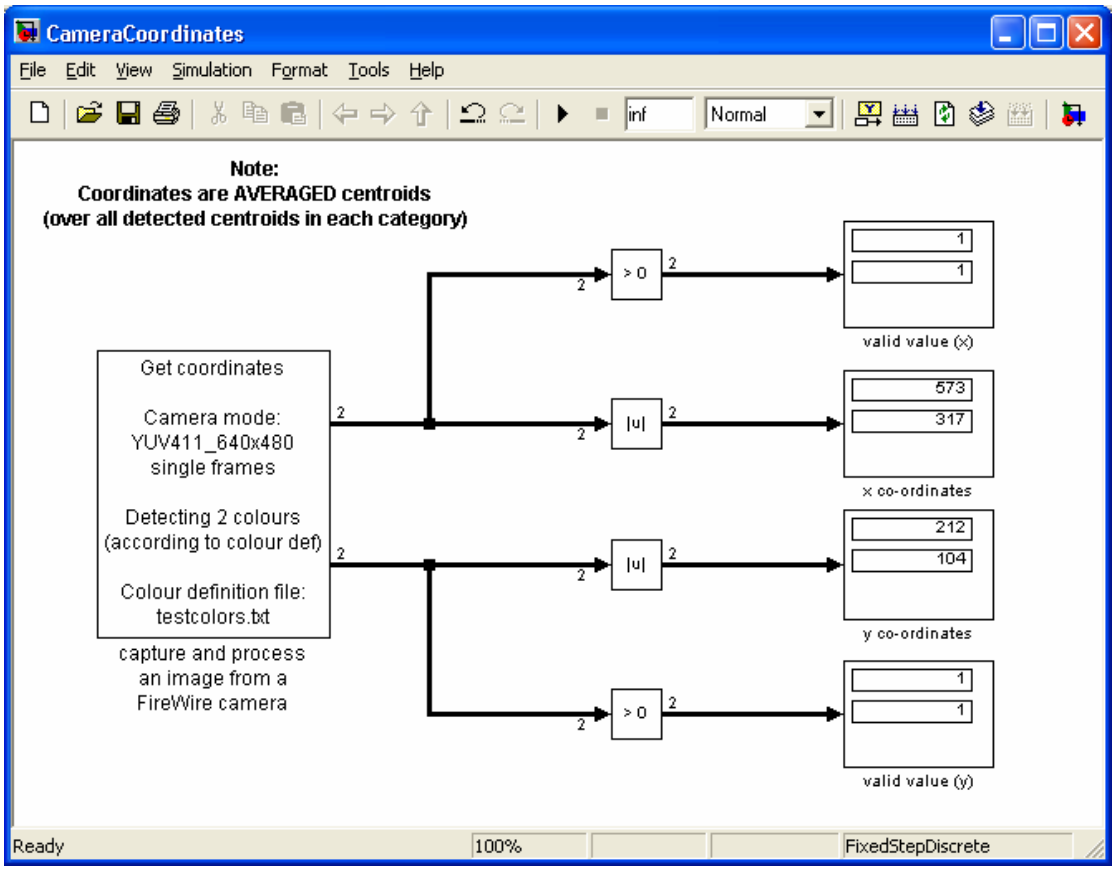
Additional information on the issue of colour training can be found in the document 'Train_UserManual.doc' which has been included in the present distribution of the toolbox.



Training the camera to 'see' the ball…

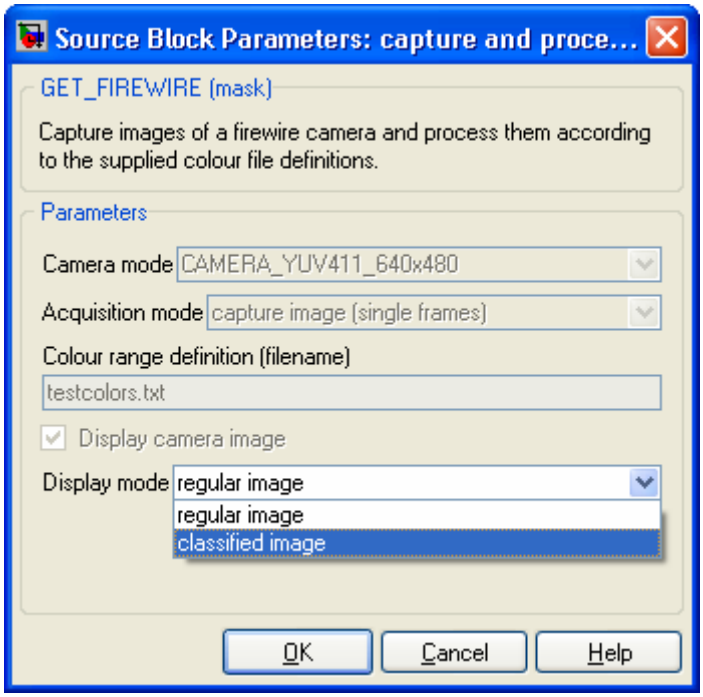
3.5 Simulink interface

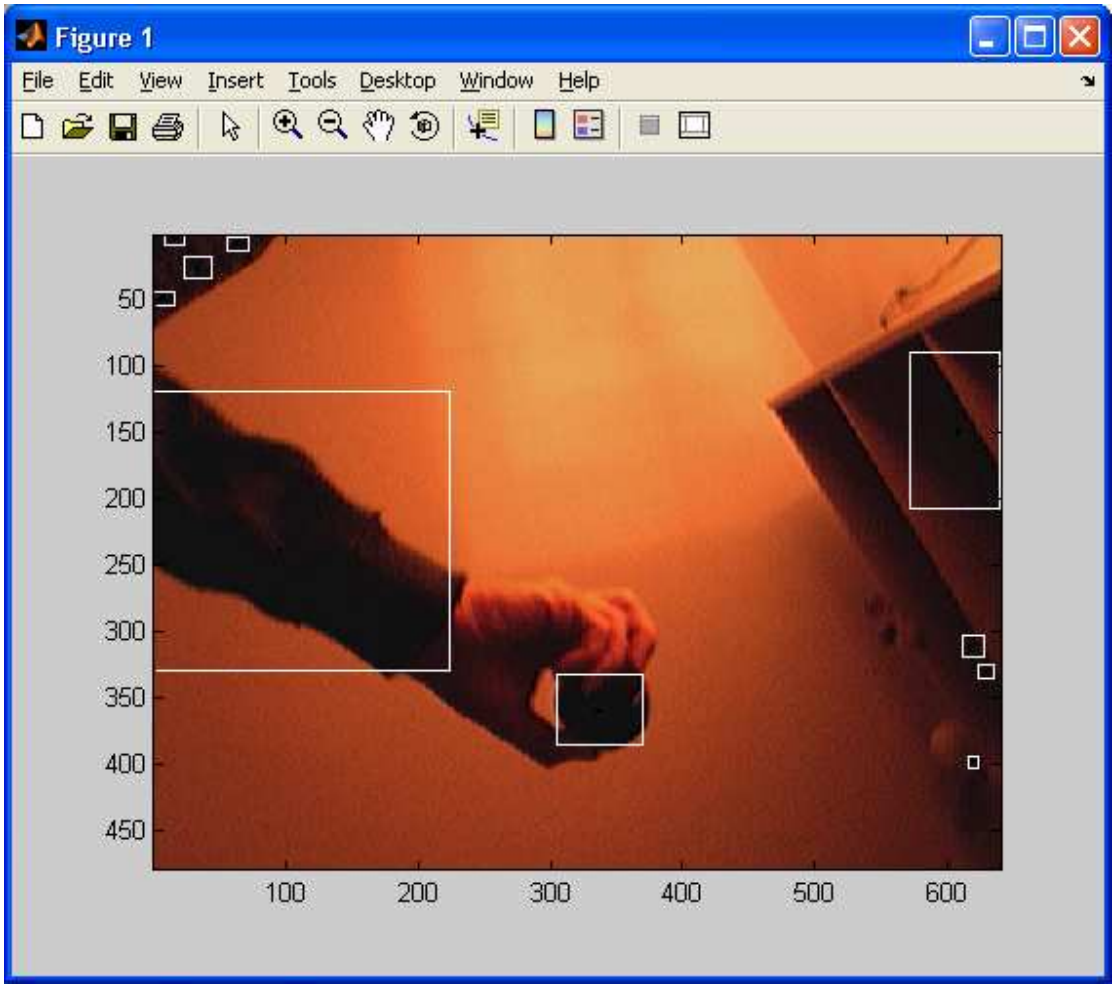
The toolbox also provides an S-Function to allow objects to be tracked from within a Simulink model (see the example in folder *SFcapProc)*. The most recent image captured by the camera is processed and the centroid coordinates (x, y) of the first detected object of each specified colour is provided as an output signal of the block.



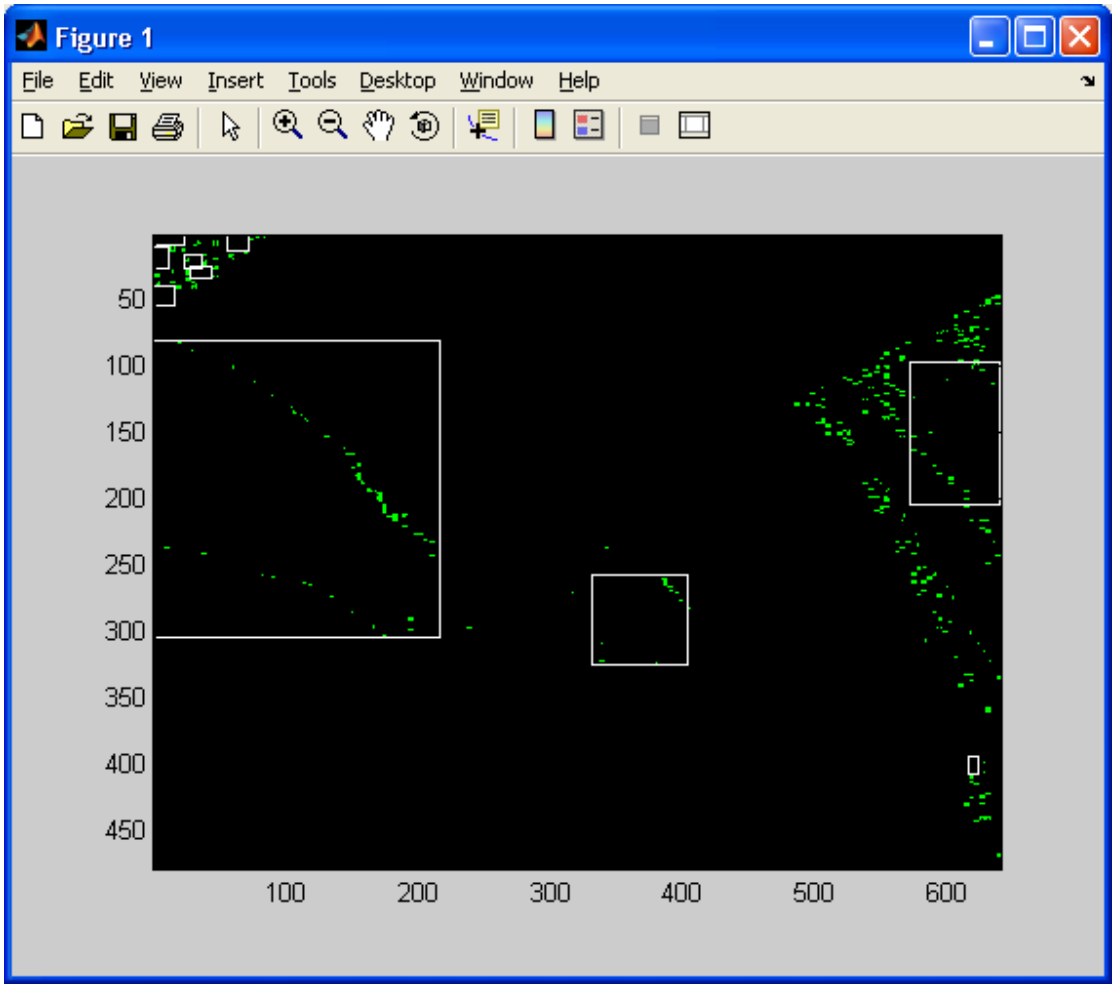
The block mask allows specification of the video mode to be used (e.g. YUV411), the acquisition mode (single frames or stream based) as well as the name of the colour definition file.

Optionally, the captured image as well as centroid information and boundary boxes can be displayed ('display' check box). When the display is enabled, the 'display format' of the image can be chosen to be a *regular image* or a *classified image* (see below).

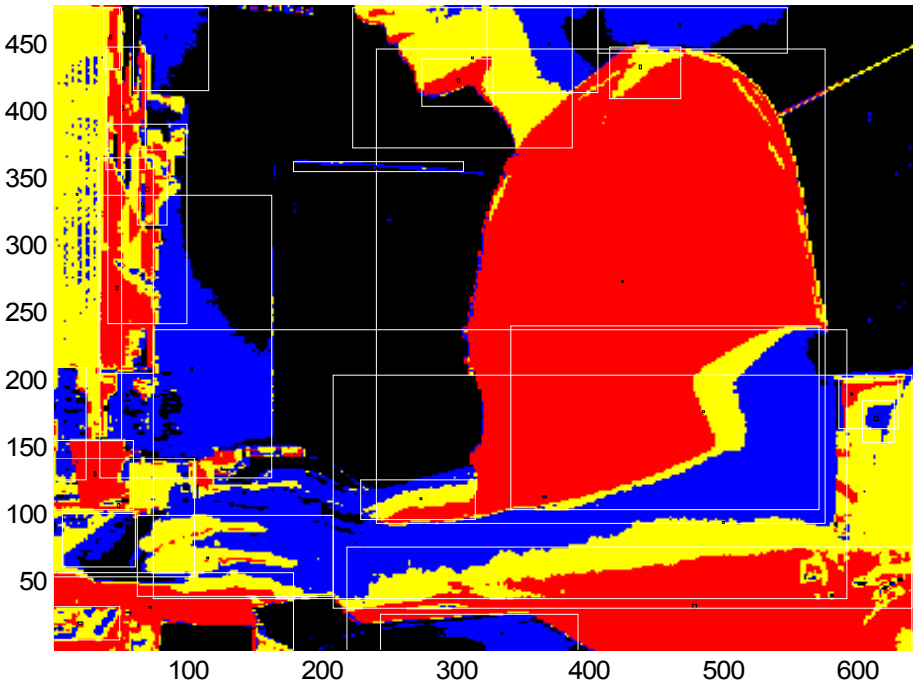




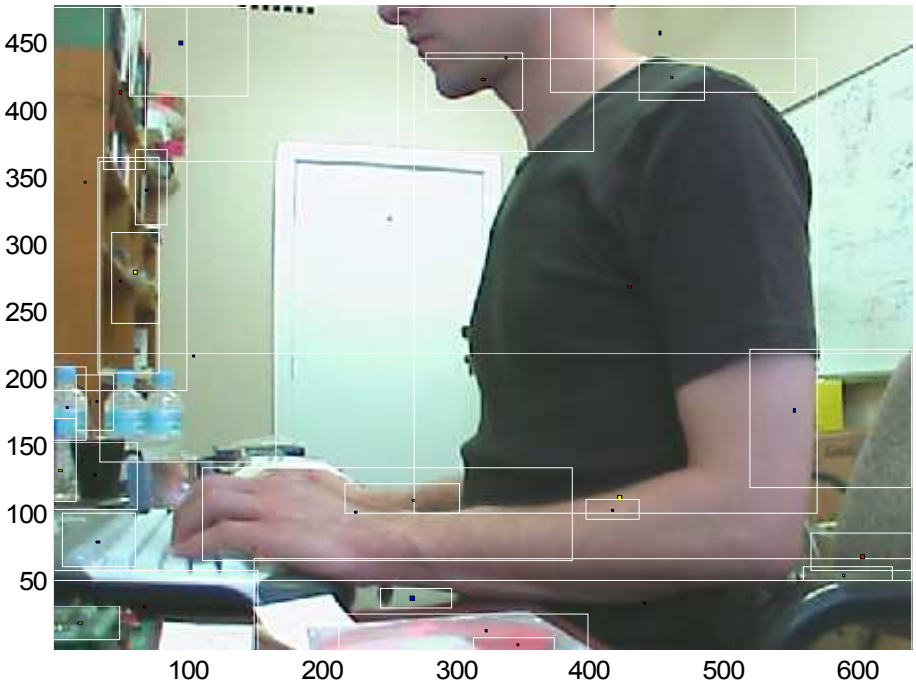
SFCapProc: Displaying a *regular image*



SFCapProc: Displaying a *classified image*



Classify mode: Detected objects are coloured, undetected objects are black



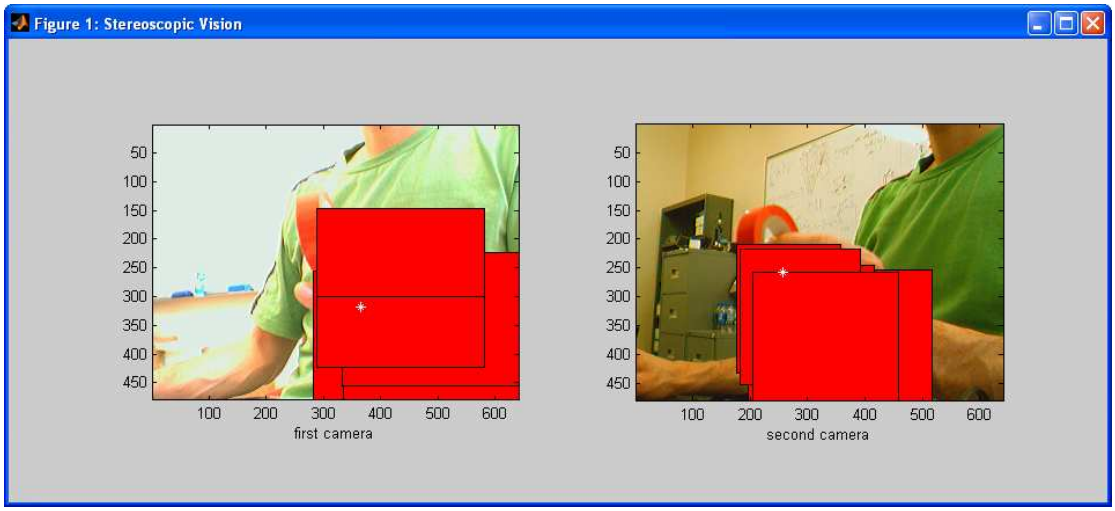
Regular display mode: Detected objects have centroids and boundary boxes

3.6 Simple stereoscopic vision

Two or more *UniBrain fire-i* cameras can be daisy-chained to transmit images on the same firewire bus interface. This can be used to provide a simple MATLAB interface for experiments with stereoscopic vision. The command *capImageStereo* assumes that two cameras have been connected to the same bus interface.

The following command sequence illustrates how two images can be captured by *capImageStereo* or captured and processed by *capProcStereo* (uses a common colour definition file) or *capProcStereo2* (uses two separate colour definition files). Note that all cameras are usually slightly different – in the figure below, the brightness property of camera 1 is larger (187) than that of camera 2 (87). The camera settings can be adjusted iteratively until one and the same colour detection specification works equally well with both cameras. Alternatively, *capProcStereo2* could be used:

```
[out1, out2, im1, im2] = capProcStereo2(2, 'c1.txt', 'c2.txt');
subplot(1,2,1)
image(im1); title('first camera'); axis image; drawnow
subplot(1,2,2)
image(im2); title('second camera'); axis image; drawnow
```



A simple stereoscopic vision system

Note that *UniBrain fire-i* cameras do not provide for synchronized sampling. Therefore, strictly speaking, this is not stereoscopic vision. However, if the objects to be tracked are not moving too fast, this method is an easy way to experiment with stereoscopic vision systems. More information about synchronized image acquisition using FireWire cameras can be found the following document:

www.mecheng.adelaide.edu.au/robotics_novell/WWW_Devs/FireWireVisionTools/FireWire_and_IIDC_cameras.pdf

Appendices

Appendix A – The camera configuration file

The camera configuration file *(cameraconfig.txt)* has the structure shown below. The ***[Format]*** section outlines the meaning of the entries for each camera. This structure is fixed and should not be modified (the driver files simply ignore this section). The entries in sections ***[Camera_1]*** and ***[Camera_2]*** define the parameters listed in the format section (in that order). The default values have been found using the configuration panel of the demo program that comes with the driver (1394CameraDemo). The values work in our indoor environment, but any slight change in the lighting conditions may require a re-adjustment.

[Format]

Shutter (6)
Saturation (90)
White Balance (96 69)
Gain (87)
Auto Exposure (511)
Brightness (304)

[Camera_1]

6
90
96 69
187
511
304

[Camera_2]

6
90
96 69
87
511
304

Appendix B – Recompiling the drivers

The file structure of this toolbox includes a 'development' folder *(_devlpmt)*. Within this folder, each of the commands has its own subfolder.

To recompile any of the commands, change directory to the corresponding subfolder, e.g.

```
>> cd _devlpmt/capProc
```

… edit the source file as required and then recompile using the provided script '**cc**':

```
>> cc
```

Once compiled, copy the newly created mex32 file (e.g. capProc.mex32) to the '_bin' folder of the toolbox. Subsequent calls to *capProc()* will now make use of the modified command.